UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/717,941	11/20/2003	Stephen La Roux Blinick	TUC920030135US1	9013

45216 7590 05/29/2008
Kunzler & McKenzie
8 EAST BROADWAY
SUITE 600
SALT LAKE CITY, UT 84111

EXAMINER	
WANG, BEN C	
ART UNIT	**PAPER NUMBER**
2192	
MAIL DATE	**DELIVERY MODE**
05/29/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No. 10/717,941	Applicant(s) BLINICK ET AL.	
	Examiner BEN C. WANG	Art Unit 2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE _3_ MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on _11 February 2008_.

2a)☒ This action is **FINAL**. 2b)□ This action is non-final.

3)□ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) _1-30_ is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)□ Claim(s) _____ is/are allowed.

6)☒ Claim(s) _1-30_ is/are rejected.

7)□ Claim(s) _____ is/are objected to.

8)□ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)□ The specification is objected to by the Examiner.

10)□ The drawing(s) filed on _____ is/are: a)□ accepted or b)□ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)□ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)□ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)□ All b)□ Some * c)□ None of:

1.□ Certified copies of the priority documents have been received.

2.□ Certified copies of the priority documents have been received in Application No. _____.

3.□ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) □ Notice of References Cited (PTO-892)

2) □ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) □ Information Disclosure Statement(s) (PTO/SB/08)
Paper No(s)/Mail Date _____.

4) □ Interview Summary (PTO-413)
Paper No(s)/Mail Date. _____ .

5) □ Notice of Informal Patent Application

6) □ Other: _____.

DETAILED ACTION

1. Applicant's amendment dated February 11, 2008, responding to the Office action dated November 9, 2007 provided in the rejection of claims 1-30, wherein claims 1, 4-8, 10-11, 18-20, 23-27, and 29-30 have been amended.

Claims 1-30 remain pending in the application and which have been fully considered by the examiner.

2. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a).

Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

3. Claims 1-4, 7, 10, 13, 20-22, 26, and 29-30 are rejected under 35 U.S.C. 103(a) as being unpatentable over Talati (Pub. No. US 2004/0044997 A1) (hereinafter 'Talati') in view of Hiller (Pat. No. US 6,658,659 B2) (hereinafter 'Hiller')

4. **As to claim 1** (Currently Amended), Talati discloses an apparatus for updating a code image (e.g., Fig. 2), comprising:

- a loader configured to load a new code image (e.g., [0014], lines 2-3; Fig. 2, element 102; [0047], lines 1-2) into a temporary memory location (e.g., Fig. 1, element 108; [0009], staging area) separate from a memory space (e.g., Fig. 1, element 110; [0009], line 2, runtime area) occupied by and used by an old code image (e.g., Fig. 2, elements 218, 110; [0046], lines 1-8; [0047], lines 1-2); and
- a copy module configured to copy the new code image into the memory space occupied by the old code image (e.g., Fig. 2, element 202; Fig. 3, element 302; [0013], lines 1-3).

But, Talati does not explicitly disclose the followings:

- a logic module configured to identify incompatibilities between the old code image and the new code image;
- a bootstrap module, within the new code image, configured to reconcile the incompatibilities.

However, in an analogous art of *compatible version module loading*, Hiller discloses the followings:

- a logic module configured to identify incompatibilities between the old code image and the new code image (e.g., Figs. 3A-3C and 5; Col. 9, Lines 22-28 – compatibility vectors allow recursive implicit program load requests … the modules identified by a first implicit load may, in turn, contain their own compatibility vectors which require implicit loading of the modules identified …); and
- a bootstrap module, within the new code image, configured to reconcile the incompatibilities (e.g., Col. 12, Lines 25-32 – To overcome this backward incompatibility (to reconcile the incompatibilities), the present invention permits (or may require) new versions of a program to restrict use by other application … allows a module to declare incompatibilities with older programs that it may interact with in order to provide a restriction on backward incompatibility. This mechanism includes a backward incompatibility expression stored in the new module)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hiller into the Talati's system to further provide the followings in the Talati system:

- a logic module configured to identify incompatibilities between the old code image and the new code image;
- a bootstrap module, within the new code image, configured to reconcile the incompatibilities.

The motivation is that the system wherein the version aware bootstrap code will check and ensure that loaded software modules are compatible with one another and will therefore execute properly as once suggested by Hiller (i.e., Abstract, Lines 10-13).

5. **As to claim 2** (original) (incorporating the rejection in claim 1), Talati discloses the apparatus wherein the old code image is updated substantially concurrent with normal execution of transactions by the apparatus (e.g., [0006]; [0008], lines 5-12]; [0011])

6. **As to claim 3** (original) (incorporating the rejection in claim 1), Talati discloses the apparatus further comprising an initialization module configured to initiate execution of a run-time segment of the new code image (e.g., [0014])

7. **As to claim 4** (Currently Amended) (incorporating the rejection in claim 1), Talati discloses the apparatus wherein the copy module copies the new code image into the memory space (e.g., Fig. 2, Copier copies new code; Fig. 3, element 302)

Hiller discloses the bootstrap module reconciles the incompatibilities (e.g., Col. 12, Lines 25-32 – To overcome this backward incompatibility (to reconcile the incompatibilities), the present invention permits (or may require) new versions of a program to restrict use by other application … allows a module to declare incompatibilities with older programs that it may interact with in order to provide a restriction on backward incompatibility. This mechanism includes a backward incompatibility expression stored in the new module)

8. **As to claim 7** (Currently Amended) (incorporating the rejection in claim 1), Talati discloses the bootstrap module is configured to reconcile incompatibilities by associating persistent data of the old code image with the new code image, such that the persistent data is available in response to execution of the run-time segment of the new code image (e.g., [0012] – the conversion module provides two separate functionalities (a) to selectively reconcile incompatibilities between the old code image and the new code image cited in claim 1; and, (b) to recognize persistent data described in this claim)

9. **As to claim 10** (Currently Amended), Talati discloses an apparatus for updating a code image (e.g., Fig. 2, copier copies new code), comprising of an update module

configured to load a new code image (e.g., [0014], lines 2-3; Fig. 2, element 102; [0047], lines 1-2) into a temporary memory location (Fig. 1, element 108; [0009], line 2, staging area) separate from a memory space occupied by and used by an old code image (e.g., Fig. 2, elements 218, 110; [0046], lines 1-8) and a bootstrap module within the new code image that executes subsequent to the update module (e.g., Fig. 2, element 202; [0047], lines 1-2)

But, Talati does not disclose the followings:

- a logic module configured to identify incompatibilities between the old code image and the new code image; and
- the bootstrap module configured to reconcile the incompatibilities prior to copying the new code image into the memory space occupied by the old code image

However, in an analogous art of *compatible version module loading*, Hiller discloses the followings:

- a logic module configured to identify incompatibilities between the old code image and the new code image (e.g., Figs. 3A-3C and 5; Col. 9, Lines 22-28 – compatibility vectors allow recursive implicit program load requests … the modules identified by a first implicit load may, in turn, contain their own compatibility vectors which require implicit loading of the modules identified …); and
- the bootstrap module configured to reconcile the incompatibilities prior to copying the new code image into the memory space occupied by the old

code image (e.g., Col. 12, Lines 25-32 – To overcome this backward incompatibility (to reconcile the incompatibilities), the present invention permits (or may require) new versions of a program to restrict use by other application … allows a module to declare incompatibilities with older programs that it may interact with in order to provide a restriction on backward incompatibility. This mechanism includes a backward incompatibility expression stored in the new module)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hiller into the Talati's system to further provide the followings in the Talati system:

- a logic module configured to identify incompatibilities between the old code image and the new code image; and
- the bootstrap module configured to reconcile the incompatibilities prior to copying the new code image into the memory space occupied by the old code image

The motivation is that the system wherein the version aware bootstrap code will check and ensure that loaded software modules are compatible with one another and will therefore execute properly as once suggested by Hiller (i.e., Abstract, Lines 10-13)

10. **As to claim 13** (original), Talati discloses a system that overlays an old code image with a new code image with minimal interruption of operations being performed by execution of the old code image (e.g., [0011]), the system comprising:

- a memory comprising an old code image (e.g., Fig. 1, element 110; Fig. 2, element 108) and a buffer (e.g., Fig. 1, element 108; Fig. 2, element 110) configured to store a new code image;
- a processor executing instructions of the old code image to perform one or more operations (e.g., Fig. 1, element 106), the processor configured to execute instructions of the old code image and the new code image (e.g., [0011]; [0032], lines 1-5);
- a data structure configured to store an old code image pointer (e.g., Fig. 2, elements 204, 208, and 212; [0023], lines 4-10) and a new code image pointer (e.g., Fig. 2, elements 206, 210, and 214; [0023], lines 11-16); wherein, in response to an interrupt, the processor begins executing bootstrap code within the new code image (e.g., [0040])

But, Talati does not disclose the bootstrap code configured to reconcile incompatibilities between the old code image and the new code image.

However, in an analogous art of compatible version module loading, Hiller discloses the followings:

- the bootstrap code configured to reconcile incompatibilities between the old code image and the new code image (e.g., Col. 12, Lines 25-32 – To overcome this backward incompatibility (to reconcile the incompatibilities), the present invention permits (or may require) <u>new versions of a program to restrict use by other application</u> … allows a module to declare incompatibilities with older programs that it may interact with in order <u>to</u>

provide a restriction on backward incompatibility. This mechanism includes a backward incompatibility expression stored in the new module)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hiller into the Talati's system to further provide the bootstrap code configured to reconcile incompatibilities between the old code image and the new code image in Talati system.

The motivation is that the system wherein the version aware bootstrap code will check and ensure that loaded software modules are compatible with one another and will therefore execute properly as once suggested by Hiller (i.e., Abstract, Lines 10-13)

11. **As to claim 20** (Currently Amended), Talati discloses a method for updating a code image (e.g., Fig. 2, Copier copies new code), comprising:

- loading a new code image into a temporary memory location (e.g., Fig. 1, element 108; [0009], staging area) separate from a memory space (e.g., Fig. 1, element 110; [0009], line 2, runtime area) occupied by and used by an old code image (e.g., Fig. 2, elements 218, 110; [0046], lines 1-8; [0047], lines 1-2);
- copying the new code image into the memory space occupied by the old code image (e.g., Fig. 2, element 202; Fig. 3, element 302).

But, Talati does not disclose the followings:

- identifying incompatibilities between the old code image and the new code images; and

- reconciling the incompatibilities using bootstrap code of the new code image.

However, in an analogous art of *compatible version module loading*, Hiller discloses the followings:

- identifying incompatibilities between the old code image and the new code images (e.g., Figs. 3A-3C and 5; Col. 9, Lines 22-28 – compatibility vectors allow recursive implicit program load requests … the modules identified by a first implicit load may, in turn, contain their own compatibility vectors which require implicit loading of the modules identified …); and
- reconciling the incompatibilities using bootstrap code of the new code image (e.g., Col. 12, Lines 25-32 – To overcome this backward incompatibility (to reconcile the incompatibilities), the present invention permits (or may require) new versions of a program to restrict use by other application … allows a module to declare incompatibilities with older programs that it may interact with in order to provide a restriction on backward incompatibility. This mechanism includes a backward incompatibility expression stored in the new module)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Talati and the teachings of Hiller to further provide the followings in the Talati system:

- identifying incompatibilities between the old code image and the new code images; and

- reconciling the incompatibilities using bootstrap code of the new code image.

The motivation is that the system wherein the version aware bootstrap code will check and ensure that loaded software modules are compatible with one another and will therefore execute properly as once suggested by Hiller (i.e., Abstract, Lines 10-13)

12. **As to claim 21** (original) (incorporating the rejection in claim 20), Talati discloses the method wherein the old code image is updated substantially concurrently with execution of regular computer operations (e.g., [0011]; [0014])

13. **As to claim 22** (original) (incorporating the rejection in claim 20), Talati discloses the method further comprising initiating execution of a run-time segment of the new code image (e.g., [0049])

14. **As to claim 26** (Currently Amended) (incorporating the rejection in claim 20), Talati discloses the method wherein reconciling the incompatibilities comprises associating the persistent data of the old code image with the new code image, such that the persistent data is available in response to execution of a run-time segment of the new code image (e.g., [0011]; [0012])

15. **As to claim 29** (Currently Amended), Talati discloses an apparatus for updating a code image (e.g., [0014], lines 2-3; Fig. 2, copier copies new code; [0047], lines 1-2), the apparatus comprising:

- means for loading a new code image (e.g., Fig. 2, element 102) into a temporary memory location (e.g., Fig. 1, element 108; [0009], staging area) separate from a memory space (e.g., Fig. 1, element 110) occupied by and used by an old code image;
- means for copying the new code image into the memory space occupied by the old code image (e.g., Fig. 2, element 202; Fig. 3, element 302)

But, Talati does not disclose the followings:

- means for identifying incompatibilities between the old code image and the new code image; and
- means for reconciling the incompatibilities using bootstrap code of the new code image.

However, in an analogous art of *compatible version module loading*, Hiller discloses the followings:

- means for identifying incompatibilities between the old code image and the new code image (e.g., Figs. 3A-3C and 5; Col. 9, Lines 22-28 – <u>compatibility vectors</u> allow recursive implicit program load requests … the modules identified by a first implicit load may, in turn, contain their own <u>compatibility vectors</u> which require implicit loading of the modules identified …); and

- means for reconciling the incompatibilities using bootstrap code of the new code image (e.g., Col. 12, Lines 25-32 – To overcome this backward incompatibility (to reconcile the incompatibilities), the present invention permits (or may require) new versions of a program to restrict use by other application … allows a module to declare incompatibilities with older programs that it may interact with in order to provide a restriction on backward incompatibility. This mechanism includes a backward incompatibility expression stored in the new module).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hiller into the Talati's system to further provide the followings in the Talati system:

- means for identifying incompatibilities between the old code image and the new code image; and
- means for reconciling the incompatibilities using bootstrap code of the new code image.

The motivation is that the system wherein the version aware bootstrap code will check and ensure that loaded software modules are compatible with one another and will therefore execute properly as once suggested by Hiller (i.e., Abstract, Lines 10-13)

16. **As to claim 30** (Currently Amended), Talati discloses an article of manufacture comprising a program storage medium readable by a processor and embodying one or

more instructions executable by a processor to perform a method for updating a code image (e.g., Fig. 1), the method comprising:

- loading a new code image into a temporary memory location separate from a memory space occupied by and used by an old code image (e.g., Fig. 2, elements 218, 110; [0046], lines 1-8; [0047], lines 1-2); and
- copying the new code image into the memory space occupied by the old code image (e.g., Fig. 2, element 202; Fig. 3, element 302)

But, Talati does not disclose the followings:

- identifying incompatibilities between the old code image and the new code image; and
- reconciling the incompatibilities using bootstrap code of the new code image

However, in an analogous art of *compatible version module loading*, Hiller discloses the followings:

- identifying incompatibilities between the old code image and the new code image (e.g., Figs. 3A-3C and 5; Col. 9, Lines 22-28 – compatibility vectors allow recursive implicit program load requests … the modules identified by a first implicit load may, in turn, contain their own compatibility vectors which require implicit loading of the modules identified …); and
- reconciling the incompatibilities using bootstrap code of the new code image (e.g., Col. 12, Lines 25-32 – To overcome this backward incompatibility (to reconcile the incompatibilities), the present invention

> permits (or may require) new versions of a program to restrict use by other application … allows a module to declare incompatibilities with older programs that it may interact with in order to provide a restriction on backward incompatibility. This mechanism includes a backward incompatibility expression stored in the new module)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hiller into the Talati's system to further provide to further provide the followings in Talati system:

- identifying incompatibilities between the old code image and the new code image; and
- reconciling the incompatibilities using bootstrap code of the new code image.

The motivation is that the system wherein the version aware bootstrap code will check and ensure that loaded software modules are compatible with one another and will therefore execute properly as once suggested by Hiller (i.e., Abstract, Lines 10-13)

17. Claim 5-6, 8-9, 11-12, 23-25, and 27-28 are rejected under 35 U.S.C. 103(a) as being unpatentable over Talati in view of Hiller and further in view of Schwabe

18. **As to claim 5** (Currently Amended) (incorporating the rejection in claim 1), Talati and Hiller do not disclose the apparatus wherein the a logic module accesses version information for the old code image and version information for the new code image to

identify an incompatibility based at least in part on a difference between the version information.

However, in an analogous art of *Remote Incremental Program Binary Compatibility Verification Using API Definitions*, Schwabe discloses the apparatus wherein the a logic module accesses version information for the old code image and version information for the new code image to identify an incompatibility based at least in part on a difference between the version information (e.g., Fig. 17, element 1440 – version; Fig. 18, element 1470 – version; Fig. 19, element 1515 – verify version of API definition file used during verification is compatible with version of referenced binary file; Fig. 20A – 3. verify backward compatible version with content; Fig. 20C – verify versions using API definitions files, elements of 1600, 1605, and 1610)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Schwabe into the Talati-Hiller's system to further provide the apparatus wherein the a logic module accesses version information for the old code image and version information for the new code image to identify an incompatibility based at least in part on a difference between the version information in the Talati-Hiller system.

The motivation is that use of the verifier enables verification of a program's integrity and allows the use of an interpreter that does not execute the usual stack monitoring instructions during program execution, thereby greatly accelerating the program interpretation process as once suggested by Schwabe (i.e., Col. 13, Line 66 through Col. 14, Line 8)

19. **As to claim 6** (Currently Amended) (incorporating the rejection in claim 5), Schwabe discloses the apparatus wherein the bootstrap module is configured to reconcile the incompatibility by updating modules that interface with the new code image (e.g., Fig. 17, element 1440 – version; Fig. 18, element 1470 – version; Fig. 19, element 1515 – verify version of API definition file used during verification is compatible with version of referenced binary file; Fig. 20A – 3. verify backward compatible version with content; Fig. 20C – verify versions using API definitions files, elements of 1600, 1605, and 1610)

20. **As to claim 8** (Currently Amended) (incorporating the rejection in claim 1), Talati and Hiller do not disclose the apparatus wherein at least one of the incompatibilities comprises different initialization requirements for the old code image and the new code image.

However, in an analogous art of *Remote Incremental Program Binary Compatibility Verification Using API Definitions*, Schwabe discloses disclose the apparatus wherein at least one of the incompatibilities comprises different initialization requirements for the old code image and the new code image (e.g., Fig. 21A, Lines 25-36)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Schwabe into the Talati-Hiller's system to further provide the apparatus wherein at least one of the incompatibilities comprises different initialization requirements for the old code image and the new code image in the Talati-Hiller system.

The motivation is use of the verifier enables verification of a program's integrity and allows the use of an interpreter that does not execute the usual stack monitoring instructions during program execution, thereby greatly accelerating the program interpretation process as once suggested by Schwabe (i.e., Col. 13, Line 66 through Col. 14, Line 8)

21. **As to claim 9** (original) (incorporating the rejection in claim 1), Talati and Hiller do not disclose the apparatus wherein at least one of the incompatibilities comprises a difference between data structures used by the old code image and data structures used by the new code image.

However, in an analogous art of *Remote Incremental Program Binary Compatibility Verification Using API Definitions*, Schwabe discloses the apparatus wherein at least one of the incompatibilities comprises a difference between data structures used by the old code image and data structures used by the new code image (e.g., Col. 22, Lines 30-33)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Schwabe into the Talati-Hiller's system to further provide the apparatus wherein at least one of the incompatibilities comprises a difference between data structures used by the old code image and data structures used by the new code image in Talati-Hiller system.

The motivation is use of the verifier enables verification of a program's integrity and allows the use of an interpreter that does not execute the usual stack monitoring instructions during program execution, thereby greatly accelerating the program interpretation process as once suggested by Schwabe (i.e., Col. 13, Line 66 through Col. 14, Line 8)

22. **As to claim 11** (original) (incorporating the rejection in claim 10), Talati and Hiller do not explicitly disclose the apparatus wherein the bootstrap module is configured to reconcile the incompatibilities base on version information for the old code image and the new code image and a copy module configured to copy the new code image over the old code image in after the incompatibilities have been reconciled.

However, in an analogous art of *Remote Incremental Program Binary Compatibility Verification Using API Definitions*, Schwabe discloses the apparatus wherein the bootstrap module is configured to reconcile the incompatibilities base on version information for the old code image and the new code image and a copy module configured to copy the new code image over the old code image in after the incompatibilities have been reconciled (e.g., Fig. 17, element 1440 – version; Fig. 18, element 1470 – version; Fig. 19, element 1515 – verify version of API definition file used during verification is compatible with version of referenced binary file; Fig. 20A – 3. verify backward compatible version with content; Fig. 20C – verify versions using API definitions files, elements of 1600, 1605, and 1610)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Schwabe into the Talati-Hiller's system to further provide the apparatus wherein the bootstrap module is configured to reconcile the incompatibilities base on version information for the old code image and the new code image and a copy module configured to copy the new code image over the old code image in after the incompatibilities have been reconciled in the Talati-Hiller system.

The motivation is use of the verifier enables verification of a program's integrity and allows the use of an interpreter that does not execute the usual stack monitoring instructions during program execution, thereby greatly accelerating the program interpretation process as once suggested by Schwabe (i.e., Col. 13, Line 66 through Col. 14, Line 8)

23. **As to claim 12** (original) (incorporating the rejection in claim 10), Talati and Hiller do not explicitly disclose the apparatus wherein at least one of the incompatibilities comprises a difference between data structures used by the old code image and data structures used by the new code image.

However, in an analogous art of *Remote Incremental Program Binary Compatibility Verification Using API Definitions*, Schwabe the apparatus wherein at least one of the incompatibilities comprises a difference between data structures used by the old code image and data structures used by the new code image (e.g., Col. 22, Lines 30-33)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Schwabe into the Talati-Hiller's system to further provide the apparatus wherein at least one of the incompatibilities comprises a difference between data structures used by the old code image and data structures used by the new code image in Talati-Hiller system.

The motivation is use of the verifier enables verification of a program's integrity and allows the use of an interpreter that does not execute the usual stack monitoring instructions during program execution, thereby greatly accelerating the program interpretation process as once suggested by Schwabe (i.e., Col. 13, Line 66 through Col. 14, Line 8)

24. **As to claim 23** (Currently Amended) (incorporating the rejection in claim 20), Talati and Hiller do not disclose the method wherein the new code image is not copied into the memory space after the incompatibilities are reconciled.

However, in an analogous art of *Remote Incremental Program Binary Compatibility Verification Using API Definitions*, Schwabe discloses the method wherein the new code image is not copied into the memory space after the incompatibilities are reconciled (e.g., Col. 5, Lines 49-53; Col. 10, Lines 40-44; Col. 11, Line 58 through Col. 12, Lines 6; Figs. 15A-15B; Col. 20, Line 64 through Col. 21, Line 4, 14-20; Fig. 17; Col. 22, Lines 4-16; Figs. 20A-20D; Col. 24, Lines 24-32)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Schwabe into the Talati-

Hiller's system to further provide the method wherein the new code image is not copied into the memory space after the incompatibilities are reconciled in the Talati-Hiller system.

The motivation is that use of the verifier enables verification of a program's integrity and allows the use of an interpreter that does not execute the usual stack monitoring instructions during program execution, thereby greatly accelerating the program interpretation process as once suggested by Schwabe (i.e., Col. 13, Line 66 through Col. 14, Line 8)

25. **As to claim 24** (Currently Amended) (incorporating the rejection in claim 20), Talati and Hiller do not disclose the method wherein identifying incompatibilities between the old code image and the new code image comprises accessing capability information for the old code image and capability information for the new code image and identifying a difference between the capability information.

However, in an analogous art of *Remote Incremental Program Binary Compatibility Verification Using API Definitions*, Schwabe discloses the method wherein identifying incompatibilities between the old code image and the new code image comprises accessing capability information for the old code image and capability information for the new code image and identifying a difference between the capability information (e.g., Col. 9, Lines 6-11)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Schwabe into the Talati-

Hiller's system to further provide the method wherein identifying incompatibilities between the old code image and the new code image comprises accessing capability information for the old code image and capability information for the new code image and identifying a difference between the capability information in the Talati-Hiller system.

The motivation is that use of the verifier enables verification of a program's integrity and allows the use of an interpreter that does not execute the usual stack monitoring instructions during program execution, thereby greatly accelerating the program interpretation process as once suggested by Schwabe (i.e., Col. 13, Line 66 through Col. 14, Line 8)

26. **As to claim 25** (Currently Amended) (incorporating the rejection in claim 24), Schwabe discloses the method wherein reconciling the incompatibilities comprises updating modules that interface with the new code image based at least in part on the difference between the capability information (e.g., Col. 9, Lines 6-11)

27. **As to claim 27** (Currently Amended) (incorporating the rejection in claim 20), Talati and Hiller do not disclose the method wherein at least one of the incompatibilities comprises different initialization requirements for the old code image and the new code image.

However, in an analogous art of *Remote Incremental Program Binary Compatibility Verification Using API Definitions*, Schwabe discloses the method wherein at least one of the incompatibilities comprises different initialization requirements for the old code image and the new code image (e.g., Fig. 21A, Lines 25-36)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Schwabe into the Talati-Hiller's system to further provide the method wherein at least one of the incompatibilities comprises different initialization requirements for the old code image and the new code image in the Talati-Hiller system.

The motivation is that use of the verifier enables verification of a program's integrity and allows the use of an interpreter that does not execute the usual stack monitoring instructions during program execution, thereby greatly accelerating the program interpretation process as once suggested by Schwabe (i.e., Col. 13, Line 66 through Col. 14, Line 8)

28. **As to claim 28** (original) (incorporating the rejection in claim 20), Talati does not disclose the method wherein at least one of the incompatibilities comprises a difference between data structures used by the old code image and data structures used by the new code image.

However, in an analogous art of *Remote Incremental Program Binary Compatibility Verification Using API Definitions*, Schwabe discloses the method wherein at least one of the incompatibilities comprises a difference between data structures used by the old code image and data structures used by the new code image (e.g., Col. 22, Lines 30-33)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Schwabe into the Talati-Hiller's system to further provide the method wherein at least one of the incompatibilities comprises a difference between data structures used by the old code image and data structures used by the new code image in Talati-Hiller system.

The motivation is that use of the verifier enables verification of a program's integrity and allows the use of an interpreter that does not execute the usual stack monitoring instructions during program execution, thereby greatly accelerating the program interpretation process as once suggested by Schwabe (i.e., Col. 13, Line 66 through Col. 14, Line 8)

29. Claim 14-19 are rejected under 35 U.S.C. 103(a) as being unpatentable over Talati in view of Schwabe and further in view of Hiller

30. **As to claim 14** (original) (incorporating the rejection in claim 13), Talati and Schwabe do not disclose the system wherein the bootstrap code overlays the new code image in memory with the old code image in response to reconciliation of the incompatibilities.

However, in an analogous art of *Compatible Version Module Loading*, Hiller discloses the system wherein the bootstrap code overlays the new code image in memory with the old code image in response to reconciliation of the incompatibilities (e.g., Fig. 2A, element 206; Fig. 2B; Col. 3, Lines 42-46; Col. 4, Lines 64-67)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Hiller into the Talati-Schwabe's system to further provide the system wherein the bootstrap code overlays the new code image in memory with the old code image in response to reconciliation of the incompatibilities in Talati-Schwabe system.

The motivation is that the system wherein the version aware bootstrap code will check and ensure that loaded software modules are compatible with one another and will therefore execute properly as once suggested by Hill (i.e., Abstract, Lines 10-13)

31. **As to claim 15** (original) (incorporating the rejection in claim 14), Talati discloses the system wherein in response to the interrupt, the processor executes an update module of the old code image that loads the new code image into the buffer (e.g., Fig. 3, step 302; [0040]; [0041])

32. **As to claim 16** (original) (incorporating the rejection in claim 15), Talati discloses the system wherein the update module stores the old code image pointer (e.g., Fig. 2, elements 204, 208, and 212; [0023], lines 4-10) and the new code image pointer (e.g., Fig. 2, elements 206, 210, and 214; [0023], lines 11-16) in the data structure.

33. **As to claim 17** (original) (incorporating the rejection in claim 16), Talati discloses the system of wherein the update module reads a new code image header identified by

the new code image pointer (e.g., Fig. 2, elements 206, 210; [0034]) to determine the location of the bootstrap code within the new code image (e.g., [0037], lines 5-7)

34. **As to claim 18** (Currently Amended) (incorporating the rejection in claim 17), Schwabe discloses the system of wherein the bootstrap code reconciles the incompatibilities by updating modules that interface with the new code image (e.g., Col. 9, Lines 6-11)

35. **As to claim 19** (Currently Amended) (incorporating the rejection in claim 18), Schwabe discloses the system wherein the bootstrap code reconciles the incompatibilities by associating persistent data of the old code image with the new code image (e.g., Fig. 17, element 1440 – version; Fig. 18, element 1470 – version; Fig. 19, element 1515 – verify version of API definition file used during verification is compatible with version of referenced binary file; Fig. 20A – 3. verify backward compatible version with content; Fig. 20C – verify versions using API definitions files, elements of 1600, 1605, and 1610)

Response to Arguments

36. Applicant’s arguments filed on February 11, 2008 have been fully considered but they are not persuasive.

In the remarks, Applicant argues that, for examples:

a) Hiller does not teach or suggest a bootstrap module to actively reconcile incompatibilities between otherwise incompatible software (recited in REMARKS, page 12, third paragraph)

Examiner's response:

a) Hiller discloses a bootstrap module to actively reconcile incompatibilities between otherwise incompatible software (e.g., (e.g., Col. 12, Lines 25-32 – To overcome this backward incompatibility (to reconcile the incompatibilities), the present invention permits (or may require) new versions of a program to restrict use by other application … allows a module to declare incompatibilities with older programs that it may interact with in order to provide a restriction on backward incompatibility. This mechanism includes a backward incompatibility expression stored in the new module)

Conclusion

37. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Ben C Wang/

Examiner, Art Unit 2192

May 20, 2008

/Tuan Q. Dam/

Supervisory Patent Examiner, Art Unit 2192